```
epoch 1, loss: 0.000457
epoch 2, loss: 0.000081
epoch 3, loss: 0.000198
```

下面我们分别比较学到的模型参数和真实的模型参数。我们从 `net` 获得需要的层，并访问其权重（`weight`）和偏差（`bias`）。学到的参数和真实的参数很接近。

```
dense = net[0]
print(true_w, dense.weight)
print(true_b, dense.bias)
```

输出：

```
[2, -3.4] tensor([[ 1.9999, -3.4005]])
4.2 tensor([4.2011])
```

小结

- 使用PyTorch可以更简洁地实现模型。
- `torch.utils.data` 模块提供了有关数据处理的工具，`torch.nn` 模块定义了大量神经网络的层，`torch.nn.init` 模块定义了各种初始化方法，`torch.optim` 模块提供了模型参数初始化的各种方法。

> 注：本节除了代码之外与原书基本相同，原书传送门

3.4 SOFTMAX回归

前几节介绍的线性回归模型适用于输出为连续值的情景。在另一类情景中，模型输出可以是一个像图像类别这样的离散值。对于这样的离散值预测问题，我们可以使用诸如softmax回归在内的分类模型。和线性回归不同，softmax回归的输出单元从一个变成了多个，且引入了softmax运算使输出更适合离散值的预测和训练。本节以softmax回归模型为例，介绍神经网络中的分类模型。

3.4.1 分类问题

让我们考虑一个简单的图像分类问题，其输入图像的高和宽均为2像素，且色彩为灰度。这样每个像素值都可以用一个标量表示。我们将图像中的4像素分别记为x_1, x_2, x_3, x_4。假设训练数据集中图像的真实标签为狗、猫或鸡（假设可以用4像素表示出这3种动物），这些标签分别对应离散值y_1, y_2, y_3。

我们通常使用离散的数值来表示类别，例如$y_1 = 1, y_2 = 2, y_3 = 3$。如此，一张图像的标签为1、2和3这3个数值中的一个。虽然我们仍然可以使用回归模型来进行建模，并将预测值就近定点化到1、2和3这3个离散值之一，但这种连续值到离散值的转化通常会影响到分类质量。因此我们一般使用更加适合离散值输出的模型来解决分类问题。

3.4.2 SOFTMAX回归模型

softmax回归跟线性回归一样将输入特征与权重做线性叠加。与线性回归的一个主要不同在于，softmax回归的输出值个数等于标签里的类别数。因为一共有4种特征和3种输出动物类别，所以权重包含12个标量（带下标的w）、偏差包含3个标量（带下标的b），且对每个输入计算o_1, o_2, o_3这3个输出：

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} + b_1, \\
o_2 &= x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} + b_2, \\
o_3 &= x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43} + b_3.
\end{aligned}
$$

图3.2用神经网络图描绘了上面的计算。softmax回归同线性回归一样，也是一个单层神经网络。由于每个输出o_1, o_2, o_3的计算都要依赖于所有的输入x_1, x_2, x_3, x_4，softmax回归的输出层也是一个全连接层。

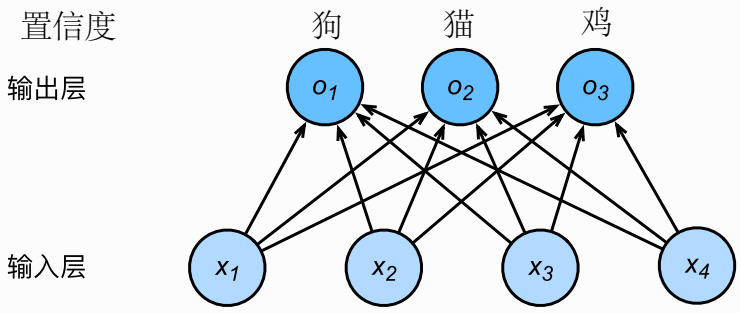


图3.2 softmax回归是一个单层神经网络

既然分类问题需要得到离散的预测输出，一个简单的办法是将输出值o_i当作预测类别是i的置信度，并将值最大的输出所对应的类作为预测输出，即输出 $\arg\max_i o_i$。例如，如果o_1, o_2, o_3分别为$0.1, 10, 0.1$，由于o_2最大，那么预测类别为2，其代表猫。

然而，直接使用输出层的输出有两个问题。一方面，由于输出层的输出值的范围不确定，我们难以直观上判断这些值的意义。例如，刚才举的例子中的输出值10表示“很置信”图像类别为猫，因为该输出值是其他两类的输出值的100倍。但如果$o_1 = o_3 = 10^3$，那么输出值10却又表示图像类别为猫的概率很低。另一方面，由于真实标签是离散值，这些离散值与不确定范围的输出值之间的误差难以衡量。

softmax运算符（softmax operator）解决了以上两个问题。它通过下式将输出值变换成值为正且和为1的概率分布：

$$\hat{y}_1, \hat{y}_2, \hat{y}_3 = \text{softmax}(o_1, o_2, o_3) \tag{16}$$

其中

$$\hat{y}_1 = \frac{\exp(o_1)}{\sum_{i=1}^{3} \exp(o_i)}, \quad \hat{y}_2 = \frac{\exp(o_2)}{\sum_{i=1}^{3} \exp(o_i)}, \quad \hat{y}_3 = \frac{\exp(o_3)}{\sum_{i=1}^{3} \exp(o_i)}. \tag{17}$$

容易看出$\hat{y}_1 + \hat{y}_2 + \hat{y}_3 = 1$且$0 \leq \hat{y}_1, \hat{y}_2, \hat{y}_3 \leq 1$，因此$\hat{y}_1, \hat{y}_2, \hat{y}_3$是一个合法的概率分布。这时候，如果$\hat{y}_2 = 0.8$，不管$\hat{y}_1$和$\hat{y}_3$的值是多少，我们都知道图像类别为猫的概率是80%。此外，我们注意到

$$\arg\max_i o_i = \arg\max_i \hat{y}_i \tag{18}$$

因此softmax运算不改变预测类别输出。

3.4.3 单样本分类的矢量计算表达式

为了提高计算效率，我们可以将单样本分类通过矢量计算来表达。在上面的图像分类问题中，假设softmax回归的权重和偏差参数分别为

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}_{4x3}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}, \tag{19}$$

设高和宽分别为2个像素的图像样本i的特征为

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & x_3^{(i)} & x_4^{(i)} \end{bmatrix}, \quad \text{i=1,2,3,...,N}$$

输出层的输出为

$$\boldsymbol{o}^{(i)} = \begin{bmatrix} o_1^{(i)} & o_2^{(i)} & o_3^{(i)} \end{bmatrix}, \quad \text{置信度}$$

预测为狗、猫或鸡的概率分布为

$$\hat{\boldsymbol{y}}^{(i)} = \begin{bmatrix} \hat{y}_1^{(i)} & \hat{y}_2^{(i)} & \hat{y}_3^{(i)} \end{bmatrix}. \quad \text{概率}$$

softmax回归对样本i分类的矢量计算表达式为

$$\begin{aligned} \boldsymbol{o}^{(i)} &= \boldsymbol{x}^{(i)} \boldsymbol{W} + \boldsymbol{b}, \\ \hat{\boldsymbol{y}}^{(i)} &= \text{softmax}(\boldsymbol{o}^{(i)}). \end{aligned}$$

3.4.4 小批量样本分类的矢量计算表达式

为了进一步提升计算效率，我们通常对小批量数据做矢量计算。广义上讲，给定一个小批量样本，其批量大小为n，输入个数（特征数）为d，输出个数（类别数）为q。设批量特征为$\boldsymbol{X} \in \mathbb{R}^{n \times d}$。假设softmax回归的权重和偏差参数分别为$\boldsymbol{W} \in \mathbb{R}^{d \times q}$和$\boldsymbol{b} \in \mathbb{R}^{1 \times q}$。softmax回归的矢量计算表达式为

$$
\begin{aligned}
\boldsymbol{O} &= \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}, \\
\hat{\boldsymbol{Y}} &= \text{softmax}(\boldsymbol{O}),
\end{aligned}
$$

其中的加法运算使用了广播机制，$\boldsymbol{O}, \hat{\boldsymbol{Y}} \in \mathbb{R}^{n \times q}$且这两个矩阵的第$i$行分别为样本$i$的输出$\boldsymbol{o}^{(i)}$和概率分布$\hat{\boldsymbol{y}}^{(i)}$。

3.4.5 交叉熵损失函数

前面提到，使用softmax运算后可以更方便地与离散标签计算误差。我们已经知道，softmax运算将输出变换成一个合法的类别预测分布。实际上，真实标签也可以用类别分布表达：对于样本i，我们构造向量$\boldsymbol{y}^{(i)} \in \mathbb{R}^{q}$，使其第$y^{(i)}$（样本$i$类别的离散数值）个元素为1，其余为0。这样我们的训练目标可以设为使预测概率分布$\hat{\boldsymbol{y}}^{(i)}$尽可能接近真实的标签概率分布$\boldsymbol{y}^{(i)}$。

我们可以像线性回归那样使用平方损失函数$\|\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}\|^2/2$。然而，想要预测分类结果正确，我们其实并不需要预测概率完全等于标签概率。例如，在图像分类的例子里，如果$y^{(i)} = 3$，那么我们只需要$\hat{y}_3^{(i)}$比其他两个预测值$\hat{y}_1^{(i)}$和$\hat{y}_2^{(i)}$大就行了。即使$\hat{y}_3^{(i)}$值为0.6，不管其他两个预测值为多少，类别预测均正确。而平方损失则过于严格，例如$\hat{y}_1^{(i)} = \hat{y}_2^{(i)} = 0.2$比$\hat{y}_1^{(i)} = 0, \hat{y}_2^{(i)} = 0.4$的损失要小很多，虽然两者都有同样正确的分类预测结果。

改善上述问题的一个方法是使用更适合衡量两个概率分布差异的测量函数。其中，交叉熵（cross entropy）是一个常用的衡量方法：

标签 预测

$$
H\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right) = -\sum_{j=1}^{q} y_j^{(i)} \log \hat{y}_j^{(i)},
$$

q为输出类别数；j为输出类别的索引；
i表示第i个样本，输入图像数目的索引。

其中带下标的$y_j^{(i)}$是向量$\boldsymbol{y}^{(i)}$中非0即1的元素，需要注意将它与样本i类别的离散数值，即不带下标的$y^{(i)}$区分。在上式中，我们知道向量$\boldsymbol{y}^{(i)}$中只有第$y^{(i)}$个元素$y_{y^{(i)}}^{(i)}$为1，其余全为0，于是$H(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) = -\log \hat{y}_{y^{(i)}}^{(i)}$。也就是说，交叉熵只关心对正确类别的预测概率，因为只要其值足够大，就可以确保分类结果正确。当然，遇到一个样本有多个标签时，例如图像里含有不止一个物体时，我们并不能做这一步简化。但即便对于这种情况，交叉熵同样只关心对图像中出现的物体类别的预测概率。

假设训练数据集的样本数为n，交叉熵损失函数定义为$\ell(\boldsymbol{\Theta}) = \frac{1}{n}\sum_{i=1}^{n} H\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right)$，

其中$\boldsymbol{\Theta}$代表模型参数。同样地，如果每个样本只有一个标签，那么交叉熵损失可以简写成$\ell(\boldsymbol{\Theta}) = -(1/n) \sum_{i=1}^{n} \log \hat{y}_{y^{(i)}}^{(i)}$。从另一个角度来看，我们知道最小化$\ell(\boldsymbol{\Theta})$等价于最大化$\exp(-n\ell(\boldsymbol{\Theta})) = \prod_{i=1}^{n} \hat{y}_{y^{(i)}}^{(i)}$，即最小化交叉熵损失函数等价于最大化训练数据集所有标签类别的联合预测概率。

3.4.6 模型预测及评价

在训练好softmax回归模型后，给定任一样本特征，就可以预测每个输出类别的概率。通常，我们把预测概率最大的类别作为输出类别。如果它与真实类别（标签）一致，说明这次预测是正确的。在3.6节的实验中，我们将使用准确率（accuracy）来评价模型的表现。它等于正确预测数量与总预测数量之比。

小结

- softmax回归适用于分类问题。它使用softmax运算输出类别的概率分布。
- softmax回归是一个单层神经网络，输出个数等于分类问题中的类别个数。
- 交叉熵适合衡量两个概率分布的差异。

> 注：本节与原书基本相同，原书此节传送门

3.5 图像分类数据集（FASHION-MNIST）

在介绍softmax回归的实现前我们先引入一个多类图像分类数据集。它将在后面的章节中被多次使用，以方便我们观察比较算法之间在模型精度和计算效率上的区别。图像分类数据集中最常用的是手写数字识别数据集MNIST[1]。但大部分模型在MNIST上的分类精度都超过了95%。为了更直观地观察算法之间的差异，我们将使用一个图像内容更加复杂的数据集Fashion-MNIST[2]（这个数据集也比较小，只有几十M，没有GPU的电脑也能吃得消）。

本节我们将使用torchvision包，它是服务于PyTorch深度学习框架的，主要用来构建计算机视觉模型。torchvision主要由以下几部分构成：

1. `torchvision.datasets`：一些加载数据的函数及常用的数据集接口；
2. `torchvision.models`：包含常用的模型结构（含预训练模型），例如AlexNet、VGG、ResNet等；
3. `torchvision.transforms`：常用的图片变换，例如裁剪、旋转等；
4. `torchvision.utils`：其他的一些有用的方法。

3.5.1 获取数据集

首先导入本节需要的包或模块。

```
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import time
import sys
sys.path.append("..") # 为了导入上层目录的d2lzh_pytorch
import d2lzh_pytorch as d2l
```

下面，我们通过torchvision的`torchvision.datasets`来下载这个数据集。第一次调用时会自动从网上获取数据。我们通过参数`train`来指定获取训练数据集或测试数据集（testing data set）。测试数据集也叫测试集（testing set），只用来评价模型的表现，并不用来训练模型。

另外我们还指定了参数`transform = transforms.ToTensor()`使所有数据转换为`Tensor`，如果不进行转换则返回的是PIL图片。`transforms.ToTensor()`将尺寸为 (H x W x C) 且数据位于[0, 255]的 PIL 图片或者数据类型为`np.uint8`的 NumPy 数组转换为尺寸为 (C x H x W) 且数据类型为`torch.float32`且位于[0.0, 1.0]的`Tensor`。

> 注意：由于像素值为0到255的整数，所以刚好是uint8所能表示的范围，包括`transforms.ToTensor()`在内的一些关于图片的函数就默认输入的是uint8型，若不是，可能不会报错但的可能得不到想要的结果。所以，**如果用像素值(0-255整数)表示图片数据，那么一律将其类型设置成uint8，避免不必要的bug。**本人就被这点坑过，详见我的这个博客2.2.4节。

```
mnist_train = torchvision.datasets.FashionMNIST(root='~/Datasets/FashionMNIST', train=True, download=True, transform=transforms.ToTensor())
mnist_test = torchvision.datasets.FashionMNIST(root='~/Datasets/FashionMNIST', train=False, download=True, transform=transforms.ToTensor())
```

上面的`mnist_train`和`mnist_test`都是`torch.utils.data.Dataset`的子类，所以我们可以用`len()`来获取该数据集的大小，还可以用下标来获取具体的一个样本。训练集中和测试集中的每个类别的图像数分别为6,000和1,000。因为有10个类别，所以训练集和测试集的样本数分别为60,000和10,000。

```
print(type(mnist_train))
print(len(mnist_train), len(mnist_test))
```

输出：

```
<class 'torchvision.datasets.mnist.FashionMNIST'>
60000 10000
```

我们可以通过下标来访问任意一个样本:

```
feature, label = mnist_train[0]
print(feature.shape, label)  # Channel x Height X Width
```

输出:

```
torch.Size([1, 28, 28]) tensor(9)
```

变量`feature`对应高和宽均为28像素的图像。由于我们使用了`transforms.ToTensor()`，所以每个像素的数值为[0.0, 1.0]的32位浮点数。需要注意的是，`feature`的尺寸是 (C x H x W) 的，而不是 (H x W x C)。第一维是通道数，因为数据集中是灰度图像，所以通道数为1。后面两维分别是图像的高和宽。

Fashion-MNIST中一共包括了10个类别，分别为t-shirt（T恤）、trouser（裤子）、pullover（套衫）、dress（连衣裙）、coat（外套）、sandal（凉鞋）、shirt（衬衫）、sneaker（运动鞋）、bag（包）和ankle boot（短靴）。以下函数可以将数值标签转成相应的文本标签。

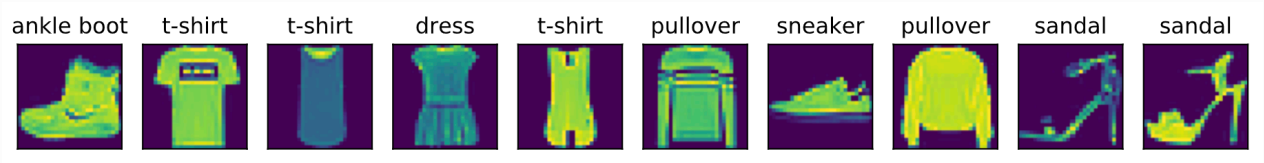
```
# 本函数已保存在d2lzh包中方便以后使用
def get_fashion_mnist_labels(labels):
    text_labels = ['t-shirt', 'trouser', 'pullover', 'dress',
'coat',
                   'sandal', 'shirt', 'sneaker', 'bag', 'ankle
boot']
    return [text_labels[int(i)] for i in labels]
```

下面定义一个可以在一行里画出多张图像和对应标签的函数。

```
# 本函数已保存在d2lzh包中方便以后使用
def show_fashion_mnist(images, labels):
    d2l.use_svg_display()
    # 这里的_表示我们忽略（不使用）的变量
    _, figs = plt.subplots(1, len(images), figsize=(12, 12))
    for f, img, lbl in zip(figs, images, labels):
        f.imshow(img.view((28, 28)).numpy())
        f.set_title(lbl)
        f.axes.get_xaxis().set_visible(False)
        f.axes.get_yaxis().set_visible(False)
    plt.show()
```

现在，我们看一下训练数据集中前9个样本的图像内容和文本标签。

```
X, y = [], []
for i in range(10):
    X.append(mnist_train[i][0])
    y.append(mnist_train[i][1])
show_fashion_mnist(X, get_fashion_mnist_labels(y))
```



3.5.2 读取小批量

我们将在训练数据集上训练模型，并将训练好的模型在测试数据集上评价模型的表现。前面说过，`mnist_train`是`torch.utils.data.Dataset`的子类，所以我们可以将其传入`torch.utils.data.DataLoader`来创建一个读取小批量数据样本的DataLoader实例。

在实践中，数据读取经常是训练的性能瓶颈，特别当模型较简单或者计算硬件性能较高时。PyTorch的`DataLoader`中一个很方便的功能是允许使用多进程来加速数据读取。这里我们通过参数`num_workers`来设置4个进程读取数据。

```
batch_size = 256
if sys.platform.startswith('win'):
    num_workers = 0  # 0表示不用额外的进程来加速读取数据
else:
    num_workers = 4
train_iter = torch.utils.data.DataLoader(mnist_train,
batch_size=batch_size, shuffle=True, num_workers=num_workers)
test_iter = torch.utils.data.DataLoader(mnist_test,
batch_size=batch_size, shuffle=False, num_workers=num_workers)
```

我们将获取并读取Fashion-MNIST数据集的逻辑封装在`d2lzh_pytorch.load_data_fashion_mnist`函数中供后面章节调用。该函数将返回`train_iter`和`test_iter`两个变量。随着本书内容的不断深入，我们会进一步改进该函数。它的完整实现将在5.6节中描述。

最后我们查看读取一遍训练数据需要的时间。

```
start = time.time()
for X, y in train_iter:
    continue
print('%.2f sec' % (time.time() - start))
```

输出：

```
1.57 sec
```

小结

- Fashion-MNIST是一个10类服饰分类数据集，之后章节里将使用它来检验不同算法的表现。
- 我们将高和宽分别为h和w像素的图像的形状记为$h \times w$或`(h, w)`。

参考文献

(1) LeCun, Y., Cortes, C., & Burges, C. http://yann.lecun.com/exdb/mnist/

(2) Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

> 注：本节除了代码之外与原书基本相同，原书传送门

3.6 SOFTMAX回归的从零开始实现

这一节我们来动手实现softmax回归。首先导入本节实现所需的包或模块。

```
import torch
import torchvision
import numpy as np
import sys
sys.path.append("..") # 为了导入上层目录的d2lzh_pytorch
import d2lzh_pytorch as d2l
```

3.6.1 获取和读取数据

我们将使用Fashion-MNIST数据集，并设置批量大小为256。

```
batch_size = 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
```

3.6.2 初始化模型参数

跟线性回归中的例子一样，我们将使用向量表示每个样本。已知每个样本输入是高和宽均为28像素的图像。模型的输入向量的长度是 $28 \times 28 = 784$：该向量的每个元素对应图像中每个像素。由于图像有10个类别，单层神经网络输出层的输出个数为10，因此softmax回归的权重和偏差参数分别为784×10和1×10的矩阵。

```
num_inputs = 784
num_outputs = 10

W = torch.tensor(np.random.normal(0, 0.01, (num_inputs,
num_outputs)), dtype=torch.float)
b = torch.zeros(num_outputs, dtype=torch.float)
```

同之前一样，我们需要模型参数梯度。

```
W.requires_grad_(requires_grad=True)
b.requires_grad_(requires_grad=True)
```

3.6.3 实现SOFTMAX运算

在介绍如何定义softmax回归之前，我们先描述一下对如何对多维 `Tensor` 按维度操作。在下面的例子中，给定一个 `Tensor` 矩阵 `X` 。我们可以只对其中同一列（ `dim=0` ）或同一行（ `dim=1` ）的元素求和，并在结果中保留行和列这两个维度（ `keepdim=True` ）。

```
X = torch.tensor([[1, 2, 3], [4, 5, 6]])
print(X.sum(dim=0, keepdim=True))
print(X.sum(dim=1, keepdim=True))
```

输出：

```
tensor([[5, 7, 9]])
tensor([[ 6],
        [15]])
```

下面我们就可以定义前面小节里介绍的softmax运算了。在下面的函数中，矩阵 `X` 的行数是样本数，列数是输出个数。为了表达样本预测各个输出的概率，softmax运算会先通过 `exp` 函数对每个元素做指数运算，再对 `exp` 矩阵同行元素求和，最后令矩阵每行各元素与该行元素之和相除。这样一来，最终得到的矩阵每行元素和为1且非负。因此，该矩阵每行都是合法的概率分布。softmax运算的输出矩阵中的任意一行元素代表了一个样本在各个输出类别上的预测概率。

```python
def softmax(X):
    X_exp = X.exp()
    partition = X_exp.sum(dim=1, keepdim=True)
    return X_exp / partition  # 这里应用了广播机制
```

可以看到，对于随机输入，我们将每个元素变成了非负数，且每一行和为1。

```python
X = torch.rand((2, 5))
X_prob = softmax(X)
print(X_prob, X_prob.sum(dim=1))
```

输出：

```
tensor([[0.2206, 0.1520, 0.1446, 0.2690, 0.2138],
        [0.1540, 0.2290, 0.1387, 0.2019, 0.2765]]) tensor([1., 1.])
```

3.6.4 定义模型

有了softmax运算，我们可以定义上节描述的softmax回归模型了。这里通过 `view` 函数将每张原始图像改成长度为 `num_inputs` 的向量。

```python
def net(X):
    return softmax(torch.mm(X.view((-1, num_inputs)), W) + b)
```

3.6.5 定义损失函数

上一节中，我们介绍了softmax回归使用的交叉熵损失函数。为了得到标签的预测概率，我们可以使用 `gather` 函数。在下面的例子中，变量 `y_hat` 是2个样本在3个类别的预测概率，变量 `y` 是这2个样本的标签类别。通过使用 `gather` 函数，我们得到了2个样本的标签的预测概率。与3.4节（softmax回归）数学表述中标签类别离散值从1开始逐一递增不同，在代码中，标签类别的离散值是从0开始逐一递增的。

0 1 2 0 1 2

```python
y_hat = torch.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y = torch.LongTensor([0, 2])
y_hat.gather(1, y.view(-1, 1))
```

输出：

```
tensor([[0.1000],
        [0.5000]])
```

下面实现了3.4节（softmax回归）中介绍的交叉熵损失函数。

```
def cross_entropy(y_hat, y):
    return - torch.log(y_hat.gather(1, y.view(-1, 1)))      P5
```

3.6.6 计算分类准确率

给定一个类别的预测概率分布`y_hat`，我们把预测概率最大的类别作为输出类别。如果它与真实类别`y`一致，说明这次预测是正确的。分类准确率即正确预测数量与总预测数量之比。

为了演示准确率的计算，下面定义准确率`accuracy`函数。其中`y_hat.argmax(dim=1)`返回矩阵`y_hat`每行中最大元素的索引，且返回结果与变量`y`形状相同。相等条件判断式`(y_hat.argmax(dim=1) == y)`是一个类型为`ByteTensor`的`Tensor`，我们用`float()`将其转换为值为0（相等为假）或1（相等为真）的浮点型`Tensor`。

item()方法把字典中每对key和value组成一个元组，并把这些元组放在列表中返回。

```
def accuracy(y_hat, y):
    return (y_hat.argmax(dim=1) == y).float().mean().item()
```

让我们继续使用在演示`gather`函数时定义的变量`y_hat`和`y`，并将它们分别作为预测概率分布和标签。可以看到，第一个样本预测类别为2（该行最大元素0.6在本行的索引为2），与真实标签0不一致；第二个样本预测类别为2（该行最大元素0.5在本行的索引为2），与真实标签2一致。因此，这两个样本上的分类准确率为0.5。

```
print(accuracy(y_hat, y))
```

输出：

```
0.5
```

类似地，我们可以评价模型`net`在数据集`data_iter`上的准确率。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用。该函数将被逐步改进：它的完整实现将在“图像增广”一节中描述
def evaluate_accuracy(data_iter, net):
    acc_sum, n = 0.0, 0
    for X, y in data_iter:
        acc_sum += (net(X).argmax(dim=1) == y).float().sum().item()
        n += y.shape[0]
    return acc_sum / n
```

因为我们随机初始化了模型`net`，所以这个随机模型的准确率应该接近于类别个数10的倒数即0.1。

```
print(evaluate_accuracy(test_iter, net))
```

输出：

```
0.0681
```

3.6.7 训练模型

训练softmax回归的实现跟"线性回归的从零开始实现"一节介绍的线性回归中的实现非常相似。我们同样使用小批量随机梯度下降来优化模型的损失函数。在训练模型时，迭代周期数`num_epochs`和学习率`lr`都是可以调的超参数。改变它们的值可能会得到分类更准确的模型。

```
num_epochs, lr = 5, 0.1

# 本函数已保存在d2lzh包中方便以后使用
def train_ch3(net, train_iter, test_iter, loss, num_epochs, batch_size,
              params=None, lr=None, optimizer=None):
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n = 0.0, 0.0, 0
        for X, y in train_iter:
            y_hat = net(X)
            l = loss(y_hat, y).sum()

            # 梯度清零
            if optimizer is not None:
                optimizer.zero_grad()
            elif params is not None and params[0].grad is not None:
                for param in params:
                    param.grad.data.zero_()

            l.backward()
            if optimizer is None:
                d2l.sgd(params, lr, batch_size)
            else:
                optimizer.step()  # "softmax回归的简洁实现"一节将用到


            train_l_sum += l.item()  损失值
            train_acc_sum += (y_hat.argmax(dim=1) == y).sum().item()  准确率
            n += y.shape[0]
        test_acc = evaluate_accuracy(test_iter, net)
        print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f'
              % (epoch + 1, train_l_sum / n, train_acc_sum / n, test_acc))

```

```
train_ch3(net, train_iter, test_iter, cross_entropy, num_epochs,
batch_size, [W, b], lr)
```

输出：

```
epoch 1, loss 0.7878, train acc 0.749, test acc 0.794
epoch 2, loss 0.5702, train acc 0.814, test acc 0.813
epoch 3, loss 0.5252, train acc 0.827, test acc 0.819
epoch 4, loss 0.5010, train acc 0.833, test acc 0.824
epoch 5, loss 0.4858, train acc 0.836, test acc 0.815
```

3.6.8 预测

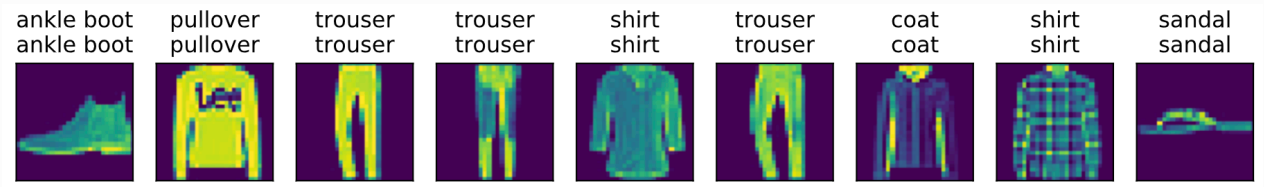
训练完成后，现在就可以演示如何对图像进行分类了。给定一系列图像（第三行图像输出），我们比较一下它们的真实标签（第一行文本输出）和模型预测结果（第二行文本输出）。

```
X, y = iter(test_iter).next()

true_labels = d2l.get_fashion_mnist_labels(y.numpy())
pred_labels =
d2l.get_fashion_mnist_labels(net(X).argmax(dim=1).numpy())
titles = [true + '\n' + pred for true, pred in zip(true_labels,
pred_labels)]

d2l.show_fashion_mnist(X[0:9], titles[0:9])
```

取出真实标签

预测结果

zip() 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表



小结

- 可以使用softmax回归做多类别分类。与训练线性回归相比，你会发现训练softmax回归的步骤和它非常相似：获取并读取数据、定义模型和损失函数并使用优化算法训练模型。事实上，绝大多数深度学习模型的训练都有着类似的步骤。

> 注：本节除了代码之外与原书基本相同，原书传送门

3.7 SOFTMAX回归的简洁实现

我们在3.3节（线性回归的简洁实现）中已经了解了使用Pytorch实现模型的便利。下面，让我们再次使用Pytorch来实现一个softmax回归模型。首先导入所需的包或模块。

```
import torch
from torch import nn
from torch.nn import init
import numpy as np
import sys
sys.path.append("..")
import d2lzh_pytorch as d2l
```

3.7.1 获取和读取数据

我们仍然使用Fashion-MNIST数据集和上一节中设置的批量大小。

```
batch_size = 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size)
```

3.7.2 定义和初始化模型

在3.4节（softmax回归）中提到，softmax回归的输出层是一个全连接层，所以我们用一个线性模块就可以了。因为前面我们数据返回的每个batch样本`x`的形状为(batch_size, 1, 28, 28), 所以我们要先用`view()`将`x`的形状转换成(batch_size, 784)才送入全连接层。

```
num_inputs = 784
num_outputs = 10

class LinearNet(nn.Module):
    def __init__(self, num_inputs, num_outputs):
        super(LinearNet, self).__init__()
        self.linear = nn.Linear(num_inputs, num_outputs)
    def forward(self, x): # x shape: (batch, 1, 28, 28)
        y = self.linear(x.view(x.shape[0], -1))
        return y

net = LinearNet(num_inputs, num_outputs)
```

我们将对 `x` 的形状转换的这个功能自定义一个 `FlattenLayer` 并记录在 `d2lzh_pytorch` 中方便后面使用。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
class FlattenLayer(nn.Module):
    def __init__(self):
        super(FlattenLayer, self).__init__()
    def forward(self, x): # x shape: (batch, *, *, ...)
        return x.view(x.shape[0], -1)
```

这样我们就可以更方便地定义我们的模型：

```
from collections import OrderedDict
net = nn.Sequential(
        # FlattenLayer(),
        # nn.Linear(num_inputs, num_outputs)
        OrderedDict([
          ('flatten', FlattenLayer()),
          ('linear', nn.Linear(num_inputs, num_outputs))])
        )
```

然后，我们使用均值为0、标准差为0.01的正态分布随机初始化模型的权重参数。

```
init.normal_(net.linear.weight, mean=0, std=0.01)  权重w的初始化
init.constant_(net.linear.bias, val=0)  偏置b的初始化
```

3.7.3 SOFTMAX和交叉熵损失函数

如果做了上一节的练习，那么你可能意识到了分开定义softmax运算和交叉熵损失函数可能会造成数值不稳定。因此，PyTorch提供了一个包括softmax运算和交叉熵损失计算的函数。它的数值稳定性更好。

```
loss = nn.CrossEntropyLoss()
```

3.7.4 定义优化算法

我们使用学习率为0.1的小批量随机梯度下降作为优化算法。

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.1)
```

3.7.5 训练模型

接下来，我们使用上一节中定义的训练函数来训练模型。

```
num_epochs = 5
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs,
batch_size, None, None, optimizer)
```

输出：

```
epoch 1, loss 0.0031, train acc 0.745, test acc 0.790
epoch 2, loss 0.0022, train acc 0.812, test acc 0.807
epoch 3, loss 0.0021, train acc 0.825, test acc 0.806
epoch 4, loss 0.0020, train acc 0.832, test acc 0.810
epoch 5, loss 0.0019, train acc 0.838, test acc 0.823
```

小结

- PyTorch提供的函数往往具有更好的数值稳定性。
- 可以使用PyTorch更简洁地实现softmax回归。

> 注：本节除了代码之外与原书基本相同，原书传送门

3.8 多层感知机

我们已经介绍了包括线性回归和softmax回归在内的单层神经网络。然而深度学习主要关注多层模型。在本节中，我们将以多层感知机（multilayer perceptron，MLP）为例，介绍多层神经网络的概念。

3.8.1 隐藏层

多层感知机在单层神经网络的基础上引入了一到多个隐藏层（hidden layer）。隐藏层位于输入层和输出层之间。图3.3展示了一个多层感知机的神经网络图，它含有一个隐藏层，该层中有5个隐藏单元。